

PLP - Primer Parcial - 1^{er} cuatrimestre de 2023

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido y número de orden en todas las hojas, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras. El orden de los ejercicios es arbitrario. Recomendamos leer el parcial completo antes de empezar a resolverlo.

Ejercicio 1 - Programación funcional

Aclaración: en este ejercicio no está permitido utilizar recursión explícita.

Se desea representar conjuntos mediante Hashing abierto (*chain addressing*). El Hashing abierto consta de dos funciones: una función de Hash, que dado un elemento devuelve un valor entero (el cual se espera que no se repita con frecuencia), y una tabla de Hash, que dado un número entero devuelve los elementos del conjunto a los que la función de Hash asignó dicho número (es decir, la preimagen de la función de Hash para ese número).

Los representaremos en Haskell de la siguiente manera:

```
data HashSet a = Hash (a -> Integer) (Integer -> [a])
```

Por contexto de uso, vamos a suponer que la tabla de Hash es una función total, que devuelve listas vacías para los números que no corresponden a elementos del conjunto. Este es un **invariante** que deberá preservarse en todas las funciones que devuelvan conjuntos.

Definir las siguientes funciones:

- `vacío :: (a -> Integer) -> HashSet a`, que devuelve un conjunto vacío con la función de Hash indicada.
- `pertenece :: Eq a => a -> HashSet a -> Bool`, que indica si un elemento pertenece a un conjunto. Es decir, si se encuentra en la lista obtenida en la tabla de Hash para el número correspondiente a la función de Hash del elemento.

Por ejemplo:

```
pertenece 5 $ agregar 1 $ agregar 2 $ agregar 1 $ vacío (flip mod 5) devuelve False.
```

```
pertenece 2 $ agregar 1 $ agregar 2 $ agregar 1 $ vacío (flip mod 5) devuelve True.
```

- `agregar :: Eq a => a -> HashSet a -> HashSet a`, que agrega un elemento a un conjunto. Si el elemento ya estaba en el conjunto, se debe devolver el conjunto sin modificaciones.
- `intersección :: Eq a => HashSet a -> HashSet a -> HashSet a` que, dados dos conjuntos, devuelve un conjunto con la misma función de Hash del primero y con los elementos que pertenecen a ambos conjuntos a la vez.
- `foldr1` (no relacionada con los conjuntos). Dar el tipo y definir la función `foldr1` para listas **sin usar recursión explícita**, recurriendo a alguno de los esquemas de recursión conocidos.

Se recomienda usar la función `error :: String -> a` para el caso de la lista vacía.

Ejercicio 2 - Cálculo Lambda Tipado

Se desea extender el Cálculo Lambda tipado con colas bidireccionales (también conocidas como *deque*).

Se extenderán los tipos y términos de la siguiente manera:

$\sigma ::= \dots \mid \text{Cola}_\sigma \quad M ::= \dots \mid \langle \rangle_\sigma \mid M \bullet M \mid \text{próximo}(M) \mid \text{desencolar}(M) \mid \text{case } M \text{ of } \langle \rangle \rightsquigarrow M; c \bullet x \rightsquigarrow M$
donde $\langle \rangle_\sigma$ es la cola vacía en la que se pueden encolar elementos de tipo σ ; $M_1 \bullet M_2$ representa el agregado del elemento M_2 al final de la cola M_1 ; los observadores $\text{próximo}(M_1)$ y $\text{desencolar}(M_1)$ devuelven, respectivamente, el primer elemento de la cola (el primero que se encoló), y la cola sin el primer elemento (estos dos últimos solo tienen sentido si la cola no es vacía); y el observador $\text{case } M_1 \text{ of } \langle \rangle \rightsquigarrow M_2; c \bullet x \rightsquigarrow M_3$ permite operar con la cola en sentido contrario, accediendo al último elemento encolado (cuyo valor se ligará a la variable x en M_3) y al resto de la cola (que se ligará a la variable c en el mismo subtérmino).

- Introducir las reglas de tipado para la extensión propuesta.
- Definir el conjunto de valores y las nuevas reglas de semántica. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia**, basta con indicar cuántas son.
Pista: puede ser necesario mirar más de un nivel de un término para saber a qué reduce. Por ejemplo, $\text{desencolar}(\langle \rangle_{\text{Nat}} \bullet 0)$ no reduce de la misma manera que $\text{desencolar}(\langle \rangle_{\text{Nat}} \bullet \underline{1} \bullet 0)$.
- Mostrar paso por paso cómo reduce la expresión: $\text{case } \langle \rangle_{\text{Nat}} \bullet \underline{1} \bullet 0 \text{ of } \langle \rangle \rightsquigarrow \text{próximo}(\langle \rangle_{\text{Bool}}); c \bullet x \rightsquigarrow \text{isZero}(x)$
- Definir como macro la función `último`, que dada una cola devuelve el último elemento que se encoló en ella. Si la cola es vacía, puede colgarse o llegar a una forma normal bien tipada que no sea un valor. Dar un juicio de tipado válido para esta función (no es necesario demostrarlo).

Ejercicio 3 - Subtipado

Se extiende el cálculo lambda con el tipo $\text{BDD}_{\sigma, \tau}$, que representa un árbol de decisiones binarias que, al evaluarse en valores de tipo σ , arrojará resultados de tipo τ . Es decir, un árbol binario no vacío cuyos nodos internos contienen predicados (funciones que devuelven valores de tipo `Bool`), los cuales se podrán evaluar en un valor dado para definir por qué rama descender hasta llegar a una hoja, la cual contiene el resultado final. ($\text{BDD} = \text{Binary Decision Diagram}$).

$\sigma ::= \dots \mid \text{BDD}_{\sigma, \tau} \quad M ::= \dots \mid \text{val}_{\sigma}(M) \mid M ? M ; M \mid \text{evaluar}(M, M)$

donde:

- $\text{BDD}_{\sigma, \tau}$ es el tipo de los diagramas de decisiones binarias que admiten valores de tipo σ y arrojan resultados de tipo τ .
- $\text{val}_{\sigma}(M)$ representa un diagrama sin predicados, que al evaluarse en un valor de tipo σ siempre arroja como resultado M - o, más precisamente, el resultado de reducir M a un valor.
- $M_1 ? M_2 ; M_3$ representa al diagrama cuya raíz es la función M_1 , y sus subárboles son M_2 y M_3 . Si al evaluar el diagrama para un valor V la aplicación de M_1 a V devuelve `True`, se continuará evaluando el subárbol M_2 . En caso contrario, se descenderá por el subárbol M_3 .
- $\text{evaluar}(M_1, M_2)$ representa la evaluación del diagrama M_1 para el argumento M_2 . Para obtener el valor de esta evaluación se deberá aplicar, en cada paso, la función del nodo actual al valor de M_2 , para luego descender por la rama que corresponda según el resultado de la aplicación sea `True` o `False`.

Las reglas de tipado son las siguientes:

$$\frac{\Gamma \triangleright M : \tau}{\Gamma \triangleright \text{val}_{\sigma}(M) : \text{BDD}_{\sigma, \tau}} (\text{T-VAL}) \quad \frac{\Gamma \triangleright M_1 : \sigma \rightarrow \text{Bool} \quad \Gamma \triangleright M_2 : \text{BDD}_{\sigma, \tau} \quad \Gamma \triangleright M_3 : \text{BDD}_{\sigma, \tau}}{\Gamma \triangleright M_1 ? M_2 ; M_3 : \text{BDD}_{\sigma, \tau}} (\text{T-CHOICE})$$

$$\frac{\Gamma \triangleright M_1 : \text{BDD}_{\sigma, \tau} \quad \Gamma \triangleright M_2 : \sigma}{\Gamma \triangleright \text{evaluar}(M_1, M_2) : \tau} (\text{T-EVAL})$$

- Dar la(s) regla(s) de subtipado para esta extensión y justificar en términos del principio de sustitutividad.
- Utilizando las reglas de tipado y subtipado, derivar el siguiente juicio de tipado:

$\emptyset \triangleright (\lambda b : \text{BDD}_{\text{Bool}, \text{Nat}}. \text{evaluar}(b, \text{True}))((\lambda y : \text{Nat}. \text{isZero}(y)) ? \text{val}_{\text{Nat}}(\text{True}) ; \text{val}_{\text{Nat}}(\text{False})) : \text{Int}$